Optimización multiobjetivo utilizando NSGA y MOACS para el problema del viajante

José Luis Benitez[1], Ivan Vera[1]
[1] Estudiantes de Ingeniería en Informática, Facultad Politécnica de la Universidad Nacional de Asunción, San Lorenzo, Paraguay
[1]{jose.luis.veron, ivera2065}@fpuna.edu.py

Resumen. El Traveling Salesman Problem (TSP) es conocido como un problema NP-Completo combinatorio que es muy importante para muchas aplicaciones en el mundo real. En este paper resolvemos un TSP multiobjetivo y hacemos una comparación entre ambos algoritmos resolviendo dos instancias del problema. Los algoritmos utilizados son el Non-dominated Sorting Genetic Algorithm (NSGA) y el Multi-Objective Ant Colony System (MOACS).

Palabras Clave: algoritmo genético, elitismo, crossover, mutación, hormigas, población, sigma share, dummy fitness, feromonas, factor de evaporación, sharing.

1 Introducción

Muchos problemas de diseño o de toma de decisiones en el mundo real implican la optimización simultánea de múltiples objetivos. En principio, la optimización multiobjetivo es muy diferente de un mono-objetivo. En la optimización mono -objetivo, se intenta obtener la mejor decisión que suele ser el mínimo o máximo global, dependiendo de si el problema de optimización es de minimización o maximización. En el caso de los multiobjetivos puede que no exista una solución que sea la mejor(mínimo o máximo global) con respecto a todos los objetivos.
En un problema típico de optimización multiobjetivo existe un conjunto de soluciones que son superiores al resto de las soluciones en el espacio de búsqueda cuando se consideran todo los objetivos, pero que son inferiores a otras soluciones en ese espacio en uno o más objetivos. Estas soluciones se conocen como el conjunto pareto óptimas o soluciones no dominadas.
Nuestro trabajo se enfoca en la implementación del No-dominated Sorting Genetic Algorithm(NSGA) y el Multi-Objective Ant Colony System (MOACS) para resolver el problema TSP Bi-objective. Luego hacemos una comparación entres algoritmos a través de las métricas M1, M2, M3 y Error para el frente pareto. Para promediar estos valores ejecutamos 5 veces cada algoritmo por cada instancia.

2 Formulación del problema

El Problema del Vendedor Viajante o conocido en inglés como Travel Salesman Problem (TSP) consiste en responder a la pregunta: Si es que tengo ***n*** ciudades todas conectadas entre sí y dada una distancia entre cada una de las ciudades, hallar el camino más corto que recorra todas las ciudades empezando y terminando en la misma. Este problema se puede utilizar por ejemplo para hallar el mejor camino para un recolector de basura que solamente debe recorrer una vez cada lugar para recogerla, también se utiliza en el diseño de circuitos electrónicos y hasta en problemas de secuenciación de ADN.

A este problema que originalmente toma solamente en cuenta las distancias se le pueden agregar otros costos como por ejemplo la calidad del camino que se recorre o dinero gastado en peajes volviéndose así un problema multiobjetivo pudiéndose utilizar así Algoritmos Evolutivos para el enfoque multiobjetivo y también los algoritmos de Colonias de Hormigas hallándose en ambos casos los correspondientes Frentes Pareto.

3 Algoritmo MOEA

El algoritmo utilizado se basa en el procedimiento de la ordenación de no dominancia, llamado Nondominated Sorting Genetic Algorithm, NSGA.
Primeramente en la selección, la población es rankeada en base a los individuos no dominados, se sigue con el proceso hasta tener toda la población clasificada. Se asigna el fitness a cada individuo en base al **dummy y sharing fitness.** Luego se reproduce la población eligiendo a los individuos con mejores fitness(elitismo) y el restante pasa por un crossover de un punto y finalmente se aplica la mutación. Se sigue con el ciclo hasta cumplirse el criterio de parada, basado en la cantidad máxima de generaciones.
Los parámetros utilizados en el algoritmo:
dummy_fitness
sigma share
tamaño población
max generaciones
porcentaje elitismo
porcentaje crossover
porcentaje mutación

Pseudocódigo: NSGA implementando en un TSP multiobjetivo.

```
program NSGA(){
```

```
inicializar_poblacion()
gen = 0
mientras gen < max_gen {
        frente = 1
        mientras ( la población no está clasificada ) {
           identificar_individuos_no_dominados()
           asignar_dummy_fitness()
           sharing_fitnes_frente_actual()
           frente = frente + 1
        }
        operador_reproducción_elitismo()
        operador_crossover()
        operador_mutación()
    gen = gen + 1
}
```

4 Algoritmo MOACO

Se usa el enfoque de Colonia de Hormigas con este algoritmo. Es el algoritmo de Multi-Objective Ant Colony System (MOACS) implementado para la solución del Problema del Vendedor Viajante con dos objetivos. Se utiliza este algoritmo en busca de minimizar los dos objetivos que se encuentran en el conjunto de datos y que se encuentre un frente con los caminos con costos minimizados. En cada iteración se mandan una cantidad **cant_horm** de hormigas en busca de minimizar estos costos actualizando las feromonas cada vez que se pasa por una arista y también actualizando cada vez que se termina un ciclo de recorrido de todas las hormigas, en dicho momento el frente pareto encontrado es el único que actualiza globalmente el valor de las feromonas.
Los parámetros utilizados e inicializados son:
$\boldsymbol{\eta_i}$ = Representa la visibilidad del objetivo i.
$\boldsymbol{\rho}$ = El factor de evaporación de las feromonas.
$\boldsymbol{\beta}$ = Impacto de la visibilidad.
$\mathbf{q_0}$ = Probabilidad de usar la regla de ir al de mayor probabilidad o utilizar una ruleta para elegir el siguiente nodo.
cant_horm = Cantidad de hormigas
Algoritmo 2. Algoritmo de MOACS implementando un TSP multiobjetivo.

```
program MOACS(){
```

```
  inicializar η1=1/costo1(i,j)
  inicializar η2=1/costo2(i,j)
  inicializar ρ,α,β,q0,colonia,cant_horm
  inicializar τ=0
  frente_pareto=construir_solucion_sin_feromonas(η1,η2)
  mientras not condicion_parada(){
    hormigas=construir_solucion(τ,η1,η2)
    frente_pareto=actualizar_frente_pareto(frente_pareto,
    hormigas)
    actualizacion_global_feromonas(τ,frente_pareto)
    generación += 1
  }
  return frente_pareto
}

function construir_solucion_sin_feromonas(η1,η2){
  caminos=[]
  for hormiga=1 to cant_horm{
    camino={}
    λ=hormiga/cant_horm
    visitados=[]
    for c=0 to cant_ciudades{
      i=camino.ultimo()
      q=random()/100
      if q<=q0{
        sig_ciudad=MAXj in Ji{η1(i,j)^(λ*β)*η2(i,j)^((1-λ)*β)}
      }else{
        for j in ciudades_por_visitar{
          prob(i,j)=(η1(i,j)^(λ*β)*η2(i,j)^((1-λ)*β))/
          SUMj in Ji(η1(i,j)^(λ*β)*η2(i,j)^((1-λ)*β))
        }
          sig_ciudad=ruleta(prob)
      }
```

```
      camino.agregar_ciudad(sig_ciudad,costo1,costo2)
      τ(i,sig_ciudad)=(1-ρ)*τ(i,sig_ciudad)+ρ*τ0
    }
    i=camino.ultimo()
    τ(i,0)=(1-ρ)*τ(i,0)+ρ*τ0
    caminos.append(camino)
  }
  return actualizar_frente_pareto(caminos)
}
function construir_solucion(τ,η1,η2,hormiga){
  caminos=[]
  for hormiga=1 to cant_horm{
    camino={}
    λ=hormiga/cant_horm
    visitados=[]
    for c=0 to cant_ciudades{
      i=camino.ultimo()
      q=random()/100
      if q<=q0{
        sig_ciudad=
        MAXj in Ji{τ(i,j)*η1(i,j)^(λ*β)*η2(i,j)^((1-λ)*β)}
      }else{
        for j in ciudades_por_visitar{
         prob(i,j)=
         (τ(i,j)*η1(i,j)^(λ*β)*η2(i,j)^((1-λ)*β))/
        SUMj in Ji(τ(i,j)*η1(i,j)^(λ*β)*η2(i,j)^((1-λ)*β))
        }
        sig_ciudad=ruleta(prob)
      }
      camino.agregar_ciudad(sig_ciudad,costo1,costo2)
      τ(i,sig_ciudad)=(1-ρ)*τ(i,sig_ciudad)+ρ*τ0
    }
    i=camino.ultimo()
```

```
    τ(i,0)=(1-ρ)*τ(i,0)+ρ*τ0
    caminos.append(camino)
  }
  return caminos
}

function actualizar_frente_pareto(frente_pareto,hormigas){
  nuevo_frente=para todo i con (i en frente_pareto,j en
  hormigas) tal que no exista j que domine a i
  return nuevo_frente
}
function actualizacion_global_feromonas(τ,frente_pareto){
  f0=Sumi in frente_pareto(i.costo1)/len(frente_pareto)
  f1=Sumi in frente_pareto(i.costo2)/len(frente_pareto)
  τ0'=1/(f0*f1)
  if τ0' > τ0 {
    τ0=τ0'
    τ=τ0
  }else{
    for h in frente_pareto{
      for i,j in h.camino{
        τ(i,j)=(1-ρ)*τ(i,j)+ρ/(costo1(i,j)*costo2(i,j))
      }
    }
  }
}
```

5 Resultados Experimentales

Los experimentos se realizaron utilizando dos instancias del problema TSP, KROAB100.tsp y KROAC100.tsp, ambas instancias cuentan con 100 ciudades y dos costos por lo tanto lo vuelven un problema bi-objetivo.
Para resolver estas instancias y evaluarlas se corrieron los algoritmos NSGA y MOACS, cada algoritmo se ejecutaba 5 veces por instancia y se calculaban las

métricas M1, M2, M3 y Error de cada frente pareto que se obtenía en cada ejecución, luego estas métricas se promediaron por cada algoritmo y por cada instancia para posteriormente poder compararlas.
El y_true fue calculado combinando todos los frentes paretos de todas las ejecuciones y dejando siempre los individuos no dominados.
Para el Sigma del M2 se utilizó el valor 5.000 porque los valores de los costos rondaban entre los 100.000 y los 200.000.

Los parámetros utilizados en MOACS para ambas instancias de TSP fueron:
ρ = 0.02
β = 0.10
$\mathbf{q_0}$ = 0.6
cant_horm = 20
Iteraciones = 250

Los parámetros utilizados en NSGA para ambas instancias de TSP fueron:
dummy_fitness = 100
sigma share = 1.7
size_poblacion = 50
max_gen = 250
porcen_elitismo = 0.08
porcen_crossover = 0.92
porcen_mutacion = 0.06

KROAB100				
Algoritmo	**M1**	**M2**	**M3**	**Error**
MOACS	12260.26	5.31	53928.42	65%
NSGA	37298.10	4.03	244.79	100%

KROAC100				
Algoritmo	**M1**	**M2**	**M3**	**Error**
MOACS	7106.43	9.63	79131.8	68%
NSGA	21598.14	4.37	249.84	100%

En el parámetro M1 se puede ver que en promedio los frentes paretos obtenidos con el MOACS se acerca mucho más al y_true que los frentes paretos obtenidos por el NSGA, esto para ambas instancias.
También con el parámetro M2 se ve que el algoritmo de MOACS distribuye mejor los individuos en el frente pareto que el NSGA, en la primera instancia se ve que no hay mucha diferencia pero para la segunda instancia si le supera más el MOACS al NSGA.
En ambos casos el NSGA tiene un 100% de error por lo tanto le supera también el MOACS.
En general el MOACS es un mejor algoritmo para el TSP porque obtiene resultados de mejor calidad según las métricas.

6 Conclusiones y trabajos futuros

Aunque existen varias técnicas clásicas de optimización multiobjetivo, estas muchas veces requieren cierta información a priori del problema. Estas implementaciones de ambos algoritmos ayuda a obtener múltiples soluciones óptimas del conjunto pareto de forma simultáneamente. Por un lado el NSGA puede mantener una población estable y una reproducción de forma uniforme entre los individuos no dominados.
Para trabajos futuros, se podrían implementar algunas mejoras al NSGA, una es de ellas es utilizar otro criterio de parada y la utilización de otro método para asignación del valor del sharing fitness, con el MOACS se podrían ir ajustando los hiperparametros iniciales para obtener mejores resultados y buscar otras condiciones de parada de las iteraciones.

Referencias

1. Barán, Benjamín; Schaerer, Matilde: A MULTIOBJECTIVE ANT COLONY SYSTEM FOR VEHICLE ROUTING PROBLEM WITH TIME WINDOWS. Centro Nacional de Computación (2003).
2. J. Horn, N. Nafploitis, and D. E. Goldberg, “A niched Pareto genetic algorithm for multiobjective optimization,” in Proceedings of the First IEEE Conference on Evolutionary Computation, Z. Michalewicz, Ed. Piscataway, NJ: IEEE Press, 1994, pp. 82–87.
3. N. Srinivas and K. Deb, “Multiobjective function optimization using nondominated sorting genetic algorithms,” Evol. Comput., vol. 2, no. 3, pp. 221–248, Fall 1995.